

Exploiting Packing Components in General-Purpose Integer Programming Solvers

Jakub Mareček

IBM Research – Ireland, B3 F14 Damastown Campus, Dublin 15

The Republic of Ireland, jakub.marecek@ie.ibm.com

December 9, 2014

Abstract

The problem of packing boxes into a large box is often a part of a larger problem. For example in furniture supply chain applications, one needs to decide what trucks to use to transport furniture between production sites and distribution centers and stores, such that the furniture fits inside. Such problems are often formulated and sometimes solved using general-purpose integer programming solvers.

This chapter studies the problem of identifying a compact formulation of the multi-dimensional packing component in a general instance of integer linear programming, reformulating it using the discretisation of Allen–Burke–Mareček, and solving the extended reformulation. Results on instances of up to 10000000 boxes are reported.

1 Introduction

It is well known that one problem may have many integer linear programming formulations, whose computational behaviour is widely different. The problem of packing three-dimensional boxes into a larger box is a particularly striking example. For the trivial problem of how many unit cubes can be packed into a $(k \times 1 \times 1)$ box, a widely known formulation of Chen/Padberg/Fasano makes it impossible to answer the question for as low a k as 12 within an hour using state of the art solvers (IBM ILOG CPLEX, FICO XPress MP, Gurobi Solver), despite the fact that (after pre-solve), the instance has only 616 rows and 253 columns and 4268 non-zeros. In contrast, a discretised (“space-indexed”) formulation of Allen et al. [2012] makes it possible to solve the instance with $k = 10000000$ within an hour, where the instance of linear programming had 10000002 rows, 10000000 columns, and 30000000 non-zeros. This is due to the fact that the discretised linear programming relaxation provides a particularly strong bound. On a large-scale benchmark of randomly generated instances,

the difference between root linear programming relaxation value and optimum to optimum has been 10.49 % and 0.37 % for the Chen/Padberg/Fasano and the formulation of Allen et al. [2012], respectively.

The Allen–Burke–Marecek formulation with adaptive discretisations is, however, often rather hard to formulate in an algebraic modelling language. First, the choice of the discretisation of the larger box is hard in any case; in terms of computational complexity, finding the best possible discretisation is Δ_2^p -Hard. Second, algebraic modelling languages such as AMPL, GAMS, MOSEL, and OPL are ill-suited to the dynamic programming required by efficient discretisation algorithms. Finally, non-uniform grids pose a major challenge in debugging the formulation and analysis of the solutions. Within the modelling language, the discretisation is hence often chosen in an ad hoc manner, without considerations of optimality.

Instead, this chapter studies the related problems of identifying a packing component in the Chen/Padberg/Fasano formulation in a larger instance of integer linear programming, automating the reformulation of Chen/Padberg/Fasano formulation into the Allen–Burke–Marecek formulation, obtaining a reasonable discretisation in the process, and translating the solutions thus obtained back to the original problem. The main contributions are:

1. An overview of integer linear programming formulations of packing boxes into a larger box, covering both the formulation of Chen/Padberg/Fasano and the discretisation of Allen et al. [2012].
2. A formal statement of the problems of extracting the component and row-block, respectively, which correspond to the formulation of Chen/Padberg/Fasano for packing boxes into a larger box, from a general integer linear programming instance. Surprisingly, we show that the extraction of the row-block is solvable by polynomial-time algorithms.
3. A formal statement of the problem of finding the best possible discretisation, i.e. reformulation of Chen/Padberg/Fasano to the formulation of Allen et al. [2012]. We show that the problem is Δ_2^p -Hard, but that there are very good heuristics.
4. A novel computational study of the algorithms for the problems above. There, we take the integer linear programming instance, extract the row-block corresponding to the Chen/Padberg/Fasano formulation, perform the discretisation, write out the integer linear programming instance using the Allen–Burke–Marecek formulation, and solve it.

A general-purpose integer linear programming solver using the above results could solve much larger instances of problems involving packing components than is currently possible, when they are formulated using the Chen/Padberg/Fasano formulation.

2 Background and Definitions

In order to motivate the study of packing components, let us consider:

Problem 1. The PRECEDENCE-CONSTRAINED SCHEDULING (PCS): Given integers $r, n \geq 1$, amounts $a_i > 1$ of resources $i = 1, 2, \dots, r$ available, resource requirements of n jobs $D \in \mathbb{R}^{n \times r}$, and p pairs of numbers $P \subseteq \{(i, j) \mid 1 \leq i < j \leq n\}$ expressing job i should be executed prior to executing j , find the largest integer k so that k jobs can be executed using the resources available.

This problem on its own has numerous important applications, notably in extraction of natural resources (Bienstock and Zuckerberg [2010], Moreno et al. [2010]), where it is known as the open-pit mine production scheduling problem, aerospace engineering (Baldi et al. [2012], Fasano and Pintér [2013]), where one needs to balance the load of an aircraft, satellite, or similar, and complex vehicle routing problems, where both weight and volume of the load is considered (e.g., Iori et al. [2007]).

Clearly, there is a packing component to Precedence-Constrained Scheduling (Problem 1). Let us fix the order of six allowable rotations in dimension three arbitrarily and define:

Problem 2. The CONTAINER LOADING PROBLEM (CLP): Given dimensions of a large box (“container”) $x, y, z > 0$ and dimensions of n small boxes $D \in \mathbb{R}^{n \times 3}$ with associated values $w \in \mathbb{R}^n$, and specification of the allowed rotations $r = \{0, 1\}^{n \times 6}$, find the greatest $k \in \mathbb{R}$ such that there is a packing of small boxes $I \subseteq \{1, 2, \dots, n\}$ into the container with value $k = \sum_{i \in I} w_i$. The packed small boxes I may be rotated in any of the allowed ways, must not overlap, and no vertex can be outside of the container.

Problem 3. The VAN LOADING PROBLEM (VLP): Given dimensions of a large box (“van”) $x, y, z > 0$, maximum mass $p \geq 0$ it can hold (“payload”), dimensions of n small boxes $D \in \mathbb{R}^{n \times 3}$ with associated values $w \in \mathbb{R}^n$, mass $m \in \mathbb{R}^n$, and specification of the allowed rotations $r = \{0, 1\}^{n \times 6}$, find the greatest $k \in \mathbb{R}$ such that there is a packing of small boxes $I \subseteq \{1, 2, \dots, n\}$ into the container with value $k = \sum_{i \in I} w_i$ and mass $\sum_{i \in I} m_i \leq p$. The packed small boxes I may be rotated in any of the allowed ways, must not overlap, and no vertex can be outside of the container.

Such problems are particularly challenging. Ever since the work of Gilmore and Gomory [1965], there has been much research on extended formulations of 2D packing problems using the notion of patterns, e.g. Madsen [1979]. See Ben Amor and Valério de Carvalho [2005] for an excellent survey. Only in the past decade or two has the attention focused to exact solvers for 3D packing problems (Martello et al. [2000]), where even the special case with rotations around combinations of axes in multiples of 90 degrees is NP-Hard to approximate (Chlebík and Chlebíková [2009]). Although there are a number of excellent heuristic solvers, the progress in exact solvers for the Container Loading Problem has been limited, so far.

Table 1: Notation used in this chapter, which matches Allen et al. [2012].

Symbol	Meaning
n	The number of boxes.
H	A fixed axis, in the set $\{X, Y, Z\}$.
α	An axis of a box, in the set $\{1, 2, 3\}$.
$L_{\alpha i}$	The length of axis α of box i .
$l_{\alpha i}$	The length of axis α of box i halved.
D_H	The length of axis H of the container.
w_i	The volume of box i in the CLP.

The Formulation of Chen/Padberg/Fasano The usual compact linear programming formulations provide only weak lower bounds. Chen et al. [1995] introduced an integer linear programming formulation using the relative placement indicator:

$$\lambda_{ij}^H = \begin{cases} 1 & \text{if box } i \text{ precedes box } j \text{ along axis } H \\ 0 & \text{otherwise} \end{cases},$$

$$\delta_{\alpha i}^H = \begin{cases} 1 & \text{if box } i \text{ is rotated so that axis } \alpha \text{ is parallel to fixed } H \\ 0 & \text{otherwise} \end{cases},$$

$$x_i^H = \text{absolute position of box } i \text{ along axis } H.$$

Using the notation of Table 1 and implicit quantification, it reads:

$$\max \sum_{i=1}^n \sum_H w_i \delta_{1i}^H \quad (1)$$

$$\text{s.t.} \quad \sum_H \delta_{2i}^H = \sum_H \delta_{1i}^H \quad (2)$$

$$\sum_H \delta_{1i}^H = \sum_{\alpha} \delta_{\alpha i}^H \quad (3)$$

$$L_{1j(i)} \lambda_{j(i)i}^H + \sum_{\alpha} l_{\alpha i} \delta_{\alpha i}^H \leq x_i^H \quad (4)$$

$$x_i^H \leq \sum_{\alpha} (D_H - l_{\alpha i}) \delta_{\alpha i}^H - L_{1j(i)} \lambda_{ij(i)}^H \quad (5)$$

$$D_H \lambda_{ji}^H + \sum_{\alpha} l_{\alpha i} \delta_{\alpha i}^H - \sum_{\alpha} (D_H - l_{\alpha j}) \delta_{\alpha j}^H \leq x_i^H - x_j^H \quad (6)$$

$$x_i^H - x_j^H \leq \sum_{\alpha} (D_H - l_{\alpha i}) \delta_{\alpha i}^H - \sum_{\alpha} l_{\alpha j} \delta_{\alpha j}^H - D_H \lambda_{ij}^H \quad (7)$$

$$\sum_H (\lambda_{ij}^H + \lambda_{ji}^H) \leq \sum_H \delta_{1i}^H \quad (8)$$

$$\sum_H (\lambda_{ij}^H + \lambda_{ji}^H) \leq \sum_H \delta_{1j}^H \quad (9)$$

$$\sum_H \delta_{1i}^H + \sum_H \delta_{1j}^H \leq 1 + \sum_H (\lambda_{ij}^H + \lambda_{ji}^H) \quad (10)$$

$$\sum_{i=1}^n \sum_H \left(\prod_{\alpha} L_{\alpha i} \right) \delta_{1i}^H \leq \prod_H D_H \quad (11)$$

$$\delta_{\alpha i}^H \in \{0, 1\}, \lambda_{ij}^H \in \{0, 1\}$$

$$L_{1i} \leq L_{2i} \leq L_{3i}, j(i) \text{ such that } L_{1j(i)} = \max\{L_{1j}\} \text{ for } 1 \leq i \neq j \leq n.$$

See Figure 1 for the sparsity pattern of a small instance, known as Pigeon-02, where at most a single unit cube out of two can be packed into a single unit cube. The constraint matrix of Pigeon-02 is 43×32 . In the figure, the column ordering is given by placing D first, δ second, λ third, and x at the end, with the highest-order-first indexing therein. D has dimension two, δ has dimension 18, λ has dimension 6 and x has dimension 6. In the figure, the row ordering is given by the order of constraints (2–11) above. Notice that a similar ordering of columns and rows is naturally produced by a parser of an algebraic modelling language, such as AMPL, GAMS, MOSEL, or OPL.

This formulation has been studied a number of times. Notably, Fasano [1999, 2004, 2008] suggested numerous improvements to the formulation. Padberg [2000] has studied properties of the formulation and, in particular, identified the subsets of constraints with the integer property. Allen et al. [2012] proposed further improvements, including symmetry-breaking constraints and means of exploitation of properties of the rotations. See Fasano [2014a,b,c] for further

extensions to Tetris-like items and further references.

Nevertheless, whilst the addition of these constraints improves the performance somewhat, the formulation remains far from satisfactory. As has been pointed out in Section 1 and can be confirmed in Table 2, Pigeon- k becomes very challenging as the number k of unit cubes to pack into $(k \times 1 \times 1)$ box grows. See Figure 2 for the sparsity patten of Pigeon-12, which is already a substantial challenge for any modern solver to date, although the constraint matrix is only 1333×552 and can be reduced to 738 rows and 288 columns in the presolve. Modern integer programming solvers fail to solve instances larger than this, even considering all the additional constraints described above.

Discretisations Discretised relaxations proved to be very strong in scheduling problems corresponding to one-dimensional packing (Sousa and Wolsey [1992], van den Akker [1994], Pan and Shi [2007]) and can be shown to be asymptotically optimal for various geometric problems both in two dimensions (Papadimitriou [1981]) and in higher (Zemel [1985]) dimensions. Beasley [1985] has extended the formulation to 2D cutting applications, in the process of deriving a non-linear formulation, for which he proposed solvers. Allen et al. [2012] have extended the formulation to the 3D problem of packing boxes into a larger box, with a considerable amount of work being done independently and subsequently Junqueira et al. [2012], de Queiroz et al. [2012], Junqueira et al. [2013]. In this formulation, the small boxes are partitioned into types $t \in \{1, 2, \dots, n\}$, where boxes of one type share the same triple of dimensions. A_t is the number of boxes of type t available. The large box is discretised into units of space, possibly non-uniformly, with the indices $(x, y, z) \in D \subset R^3$ of used to index the space-indexed binary variable:

$$\mu_{x,y,z}^t = \begin{cases} 1 & \text{if a box of type } t \text{ is placed such that its lower-back-left} \\ & \text{vertex is at coordinates } x, y, z \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

Without allowing for rotations, the formulation reads:

Figure 1: The A matrix corresponding to the previously unsolved instance Pigeon-02 in the Chen/Padberg/Fasano formulation, with colour highlighting the absolute value of the coefficients.

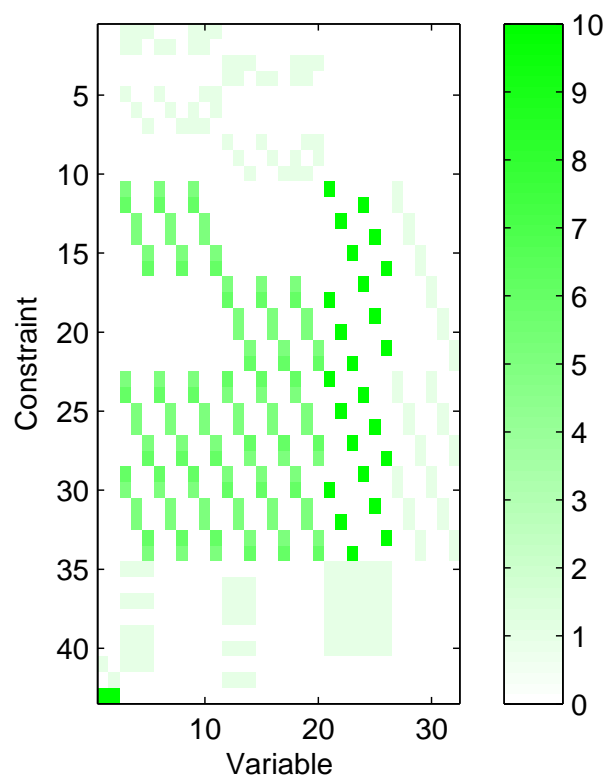
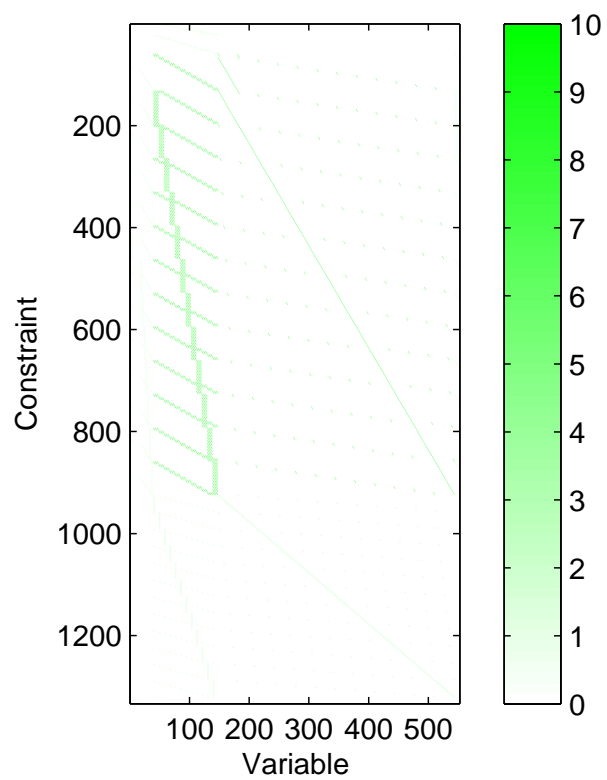


Figure 2: The A matrix corresponding to instance Pigeon-12 in the Chen/Padberg/Fasano formulation, with colour highlighting the absolute value of the coefficients.



$$\max \sum_{x,y,z,t} \mu_{xyz}^t w_t \quad (13)$$

$$\text{s.t. } \sum_t \mu_{xyz}^t \leq 1 \quad \forall x, y, z \quad (14)$$

$$\mu_{xyz}^t = 0 \quad \forall x, y, z, t \text{ where } x + L_{1t} > D_X \text{ or} \\ \text{or } y + L_{2t} > D_Y \text{ or } z + L_{3t} > D_Z \quad (15)$$

$$\sum_{x,y,z} \mu_{xyz}^t \leq A_t \quad \forall t \quad (16)$$

$$\sum_{x',y',z',t' \in f(D,n,L,x,y,z,t)} \mu_{x'y'z'}^{t'} \leq 1 \quad \forall x, y, z, t \quad (17)$$

$$\mu_{xyz}^t \in \{0, 1\} \quad \forall x, y, z, t \quad (18)$$

where one may use Algorithm 1 or similar¹ to generate the index set f in Constraint 17.

The constraints are very natural: No region in space may be occupied by more than one box type (14), boxes must be fully contained within the container (15), there may not be more than A_t boxes of type t (16), and boxes cannot overlap (17). There is one non-overlapping constraint (17) for each discretised unit of space and type of box.

In order to support rotations, new box types need to be generated for each allowed rotation and linked via set packing constraints, which are similar to Constraint 16. In order to extend the formulation to the van loading problem, it suffices to add the payload capacity constraint $\sum_{x,y,z,t} \mu_{xyz}^t m_t \leq p$.

3 Finding the Precedence-Constrained Component

In the rest of the chapter, our goal is to extract the precedence-constrained component in Chen/Padberg/Fasano formulation from a general integer linear program, and reformulate it into the discretised formulation. First, let us state the problem of extracting the Chen/Padberg/Fasano relaxation formally:

Problem 4. PRECEDENCE-CONSTRAINED COMPONENT/ROW-BLOCK EXTRACTION: Given positive integers d, m_1, m_2 , an $m_1 \times d$ integer matrix A_1 , an $m_2 \times d$ integer matrix A_2 , an m_1 -vector b_1 of integers, and an m_2 -vector b_2 of integers, corresponding to a mixed integer linear program with constraints $A_1 y = b_1$, $A_2 y \leq b_2$, find the largest integer n (“the maximum number of boxes”), such that:

¹ For one particularly efficient version of Algorithm 1, see class `DiscretisedModelSolver` and its method `addPositioningConstraints` in file `DiscretisedModelSolver.cpp` available at <http://discretisation.sourceforge.net/spaceindexed.zip> (September 30th, 2014). This code is distributed under GNU Lesser General Public License.

Algorithm 1 $f(D, n, L, x, y, z, t)$

```

1: Input: Discretisation as indices  $D \subset R^3$  of  $\mu$  variables (12), number of
   boxes  $n$ , dimensions  $L_{\alpha i} \in R \ \forall \alpha = x, y, z, i = 1, \dots, n$ , indices  $(x, y, z) \in$ 
    $D$ ,  $t \in \{1, 2, \dots, n\}$  of one scalar within  $\mu$  (12)
2: Output: Set of indices of  $\mu$  variables (12) to include in a set-packing in-
   equality (17)
3: Set  $S \leftarrow \{(x, y, z, t)\}$ 
4: for each other unit  $(x', y', z') \in D, (x, y, z) \neq (x', y', z')$  do
5:   for each box type  $t' \in \{1, 2, \dots, n\}$  do
6:     if box of type  $t$  at  $(x, y, z)$  overlaps box of type  $t'$  at  $(x', y', z')$ , i.e.,
        $(x \leq x' + L_{x't'} \leq x + L_{xt}) \wedge (y \leq y' + L_{y't'} \leq y + L_{yt})$ 
        $\wedge (z \leq z' + L_{z't'} \leq z + L_{zt})$  then
7:        $S \leftarrow S \cup \{(x', y', z', t')\}$ 
8:     end if
9:   end for
10: end for
11: return  $S$ 

```

- there exists a $4n \times 9n$ submatrix/row-block E of A_1 , corresponding only to binary variables, which we denote δ , with zero coefficients elsewhere in the rows
- there exists a $9n(n-1)/2 + 6n + 1 \times 3n(n-1)/2 + 12n$ submatrix/row-block F of A_2 , corresponding to $9n$ binary variables δ as before, $3n(n-1)/2$ binary variables denoted λ , and $3n$ continuous variables denoted x , with zero coefficients elsewhere in the rows
- E contains n rows with exactly 2 non-zero coefficients ± 1 , corresponding to (2), and 0 in the right-hand side b_1
- E contains $3n$ rows with exactly 4 non-zero coefficients ± 1 , corresponding to (3), and 0 in the right-hand side b_1
- F contains $6n$ rows with exactly 5 non-zero coefficients, some not necessarily ± 1 , corresponding to (4–5), and 0 in the right-hand side b_2
- F contains $6n(n-1)$ rows with exactly 9 non-zero coefficients, some not necessarily ± 1 , corresponding to (6–7), and 0 in the right-hand side b_2
- F contains $2n(n-1)$ rows with exactly 9 non-zero coefficients ± 1 , corresponding to (8–9), and 0 in the right-hand side b_2
- F contains $n(n-1)$ rows with exactly 12 non-zero coefficients ± 1 , corresponding to (10), and 1 in the right-hand side b_2
- F contains 1 rows with exactly $3n$ non-zero coefficients, not necessarily ± 1 , corresponding to (11), and a positive number in the right-hand side b_2 .

Algorithm 2 PrecedenceConstrainedBlock(A_1, b_1, A_2, b_2)

```
1: Input:  $A_1x = 1, A_2x \leq 1$ , that is  $m_1 \times d$  matrix  $A_1$  and  $m_2 \times d$  matrix  $A_2$ ,  
    $m_1$ -vector  $b_1$ ,  $m_2$ -vector  $b_2$   
2: Output: Integer  $k$  and blocks  $E, F$  of  $A_1, A_2$   
3: Set  $k_{\max}$  to the largest integer  $k$  such that there are  $k$  subsequent rows in  
    $A_1$  with exactly 6 non-zero elements, all  $\pm 1$   
4: for integer  $k = 4n$  from  $k_{\max}$  down to 4 do  
5:   for  $4n \times 9n$  block  $E$  in  $A_1$  such that all rows have 6 non-zeros  $\pm 1$  do  
6:     if there are no 6n other rows  $A_1$  in corresponding to (3) then  
7:       Continue  
8:     end if  
9:     for  $7n + 9n(n-1)/2 \times 3n^2 + 9n$  block  $F$  in  $A_2$  do  
10:      if  $F$  cannot be partitioned into (4-11) then  
11:        Continue  
12:      end if  
13:      return  $n, E, F$   
14:    end for  
15:  end for  
16: end for
```

Notice that by maximising the number of rows involved, we also maximise the number of boxes, as number $r = 9n(n-1)/2 + 6n + 1$ of rows is determined by number n of boxes. The following can be seen easily:

Theorem. PRECEDENCE-CONSTRAINED ROW-BLOCK EXTRACTION is in \mathcal{P} .

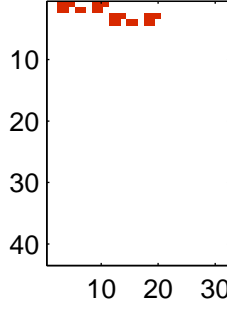
Proof sketch. A polynomial-time algorithm for extracting the precedence-constrained block can clearly rely on there being a polynomial number of blocks of the required size. See Algorithm 2. \square

Algorithm 2 displays a very general algorithm schema for PRECEDENCE-CONSTRAINED ROW-BLOCK EXTRACTION. Notably, the test of Line 10 requires elaboration. First, one needs to partition the block into the 5 families of rows (4-11). Some rows (8-9, 4-5, 11) are clearly determined by the numbers of non-zeros (9, 5, and 3). One can distinguish between others (6-7 and 10) by their right-hand sides. The test as to whether the rows represent the constraints (4-11) is based on identifying the variables. Continuous variables x are, however, identified easily and binary variables δ are determined in Line 6. What remains are variables λ .

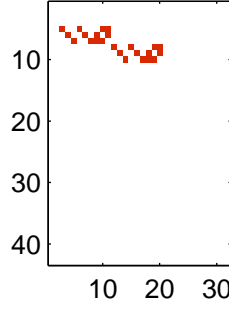
4 Exploiting the Packing Component

In order to reduce the number of regions of space, and thus the number of variables in the formulation, a sensible space-discretisation method should be employed. In many transport applications, for instance, there are only a small

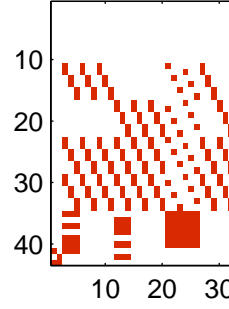
Figure 3: The workings of Algorithm 1 illustrated on instance Pigeon-02 in the Chen/Padberg/Fasano formulation, as introduced in Figure 1.



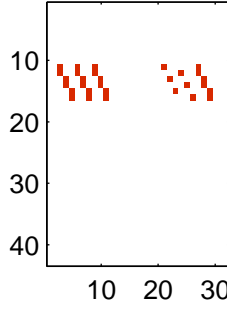
Line 5: Block E with equalities (2)



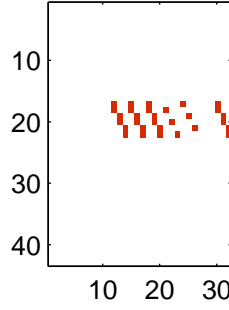
Line 6: The remaining equalities (3)



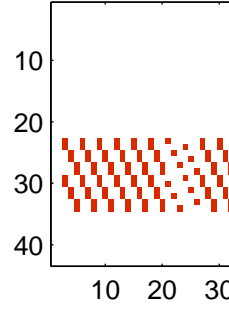
Line 9: The remaining inequalities (4-10)



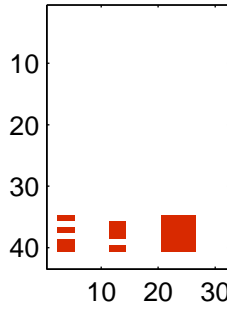
Inequalities (4)



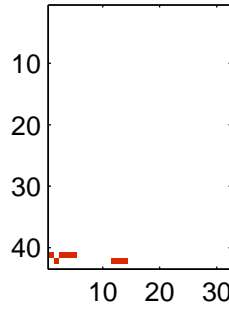
Inequalities (5)



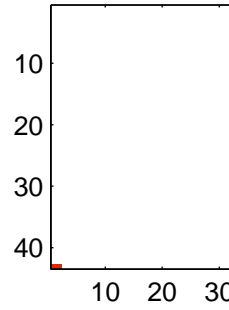
Inequalities (6-7)



Inequalities (8-9)



Inequality (10)



Inequality (11)

number of package types, with the ISO 269 standard giving the dimensions of the package. Using the discretisation of one millimeter, one could indeed introduce $162 \times 229 \times h$ variables to represent a package with the C5 base and height h millimeters, but if there are only packages with base-sizes specified by ISO 216 standard and larger than C5 to be packed in the batch, it would make sense to discretise to units of space representing 162 x 229 mm in two dimensions. The question is how to derive such a discretisation in a general-purpose system.

The greatest common divisor (GCD) reduction can be applied on a per-axis basis, finding the greatest common divisor between the length of the container for an axis and all the valid lengths of boxes that can be aligned along that axis and scaling by the inverse of the GCD. This is trivial to do and is useful when all lengths are multiples of a large number, which may be common in certain situations.

In other situations, this may not reduce the number of variables at all, and it may be worth tackling the optimisation variant of:

Problem 5. The DISCRETISATION DECISION: Given integer k , dimensions of a large box (“container”) $x, y, z > 0$ and dimensions of n small boxes $D \in \mathbb{R}^{n \times 3}$ with associated values $w \in \mathbb{R}^n$, and specification $r = \{0, 1\}^{n \times 6}$ of what rotations are allowed, decide if there are k points, where the lower-bottom-left vertex of each box can be positioned in any optimum solution of CONTAINER LOADING PROBLEM. Formally, $k = |I_X||I_Y||I_Z|$, where index-sets I_X, I_Y, I_Z refer to the set $\{1c, 2c, \dots, \max\{x, y, z\}c^{-1}\}$, where c is the greatest common divisor across elements in D .

Considering the following:

Theorem (Papadimitriou [1984]). The decision whether the optimum of an instance of KNAPSACK is unique is Δ_2^P -Complete, where Δ_2^P is the class of problems that can be solved in polynomial time using oracles from \mathcal{NP} .

Theorem. DISCRETISATION DECISION is Δ_2^P -Hard.

Proof sketch. One could check for the uniqueness of the optimum of an instance of KNAPSACK using any algorithm for DISCRETISATION DECISION (Problem 5). \square

We can, however, use non-trivial non-linear space-discretisation heuristics. Early examples include Herz [1972]. We use the same values as before on a per-axis basis, i.e. the lengths of any box sides that can be aligned along the axis. We then use dynamic programming to generate all valid locations for a box to be placed. See Algorithm 3. For example, given an axis of length 10 and box lengths of 3, 4 and 6, we can place boxes at positions 0, 3, 4, 6, and 7. 8, 9 and 10 are also possible, but no length is small enough to still lie within the container if placed at these points. This has reduced the number of regions along that axis from 10 to 5. An improvement on this scale may not be particularly common in practice, though the approach can help where the GCD is 1. It is obvious that this approach can be no worse than the GCD method

Algorithm 3 DiscretisationDP(M, n, D, r)

- 1: **Input:** Dimensions $M \in \mathbb{R}^3$ of the container, dimensions of n small boxes $D \in \mathbb{R}^{n \times 3}$, and allowed rotations $r = \{0, 1\}^{n \times 6}$
 - 2: **Output:** Integer k and k possible positions
 - 3: $P = \emptyset$
 - 4: **for** axis with limit $m \in M$ **do**
 - 5: $L = \{d \mid d \in D \text{ may appear along this axis, given allowed rotations } r\}$
 - 6: $P = \text{closure of } P \cup \{p + l \mid p \in P, l \in L, p + l \leq m\}$,
 optionally pruning $p \in P$ that cannot occur due to the dependence of the
 axes and the fact each box can be packed at most once
 - 7: **end for**
 - 8: **return** $|P|, P$
-

at discretising the container. This also adds some implicit symmetry breaking into the model. Notice that the algorithm runs in time polynomial in the size of the output it produces, but this is may be exponential in the size of the input and considerably more than the size of the best possible output.

5 Computational Experience

The formulations were tested on two sets of instances, introduced in Allen et al. [2012]:

- 3D Pigeon Hole Problem instances, Pigeon- n , where $n + 1$ unit cubes are to be packed into a container of dimensions $(1 + \epsilon) \times (1 + \epsilon) \times n$.
- SA and SAX datasets, which are used to test the dependence of solvers' performance on parameters of the instances, notably the number of boxes, heterogeneity of the boxes, and physical dimensions of the container. There is 1 pseudo-randomly generated instance for every combination of container sizes ranging from 5–100 in steps of 5 units cubed and the number of boxes to pack ranging from 5–100 in steps of 5. The SA datasets are perfectly packable, i.e. are guaranteed to be possible to load the container with 100% utilisation with all boxes packed. The SAX are similar but have no such guarantees; the summed volume of the boxes is greater than that of the container.

All of the instances are available at <http://discretisation.sf.net>. Some of these instances have been included in MIPLIB 2010 by Koch et al. [2011] and have been widely utilised in benchmarking of integer programming solvers ever since.

For the 3D Pigeon Hole Problem, results obtained within one hour using three leading solvers and the Chen/Padberg/Fasano formulation without any reformulation are shown in Table 2, while Table 3 compares the results on both

Table 2: The performance of various solvers on 3D Pigeon Hole Problem instances encoded in the Chen/Padberg/Fasano formulation. “–” denotes that optimality of the incumbent solution has not been proven within an hour.

	Time (s)		
	Gurobi 4.0	CPLEX 12.4	SCIP 2.0.1 + CLP
Pigeon-01	< 1	< 1	< 1
Pigeon-02	< 1	< 1	< 1
Pigeon-03	< 1	< 1	< 1
Pigeon-04	< 1	< 1	< 1
Pigeon-05	< 1	< 1	3.3
Pigeon-06	< 1	< 1	37.9
Pigeon-07	1.5	< 1	779.3
Pigeon-08	7.4	< 1	–
Pigeon-09	88.6	66.4	–
Pigeon-10	1381.4	686.3	–
Pigeon-11	–	–	–
Pigeon-12	–	–	–

formulations. These tests and further tests reported below were performed on a 64-bit computer running Linux, which was equipped with 2 quad-core processors (Intel Xeon E5472) and 16 GB memory. The solvers tested were IBM ILOG CPLEX 12.4, Gurobi Solver 4.0, and SCIP 2.0.1 of Achterberg [2009] with CLP as the linear programming solver. Pigeon-02 is easy to solve for any modern solver. IBM ILOG CPLEX 12.4 eliminates 17 rows and 26 columns in presolve and performs a number of further changes. The reduced instance has 23 rows and 22 columns and the reported run-time is 0.00 seconds. Pigeon-10 is the largest instance reliably solvable within an hour, but that should not be surprising, considering it has 525 rows, 220 columns, and 3600 nonzeros in the constraint matrix after presolve of CPLEX 12.4. None of the solvers managed to prove optimality of the incumbent solution for Pigeon-12 within an hour using the Chen/Padberg/Fasano formulation, although the instance of linear programming had only 627 rows, 253 columns, and 4268 non-zeros after pre-solve. As of September 2014, instances up to pigeon-13 using the Chen/Padberg/Fasano, have been solved in the process of testing integer programming solvers without the automatic reformulation, albeit at a great expense of computing time. In contrast, the reformulation and discretisation makes it possible to solve Pigeon-10000000 within an hour, where the instance of linear programming had 10000002 rows, 10000000 columns, and 30000000 non-zeros. The time for the extraction and reformulation of the instance was under 1 second across of the instances.

Figure 4: The best solutions obtained within an hour per solver per instance from the SA dataset for varying number of boxes (vertical axis) and the length of the side of the container (horizontal axis). The colours highlight the volume utilisation in percent.

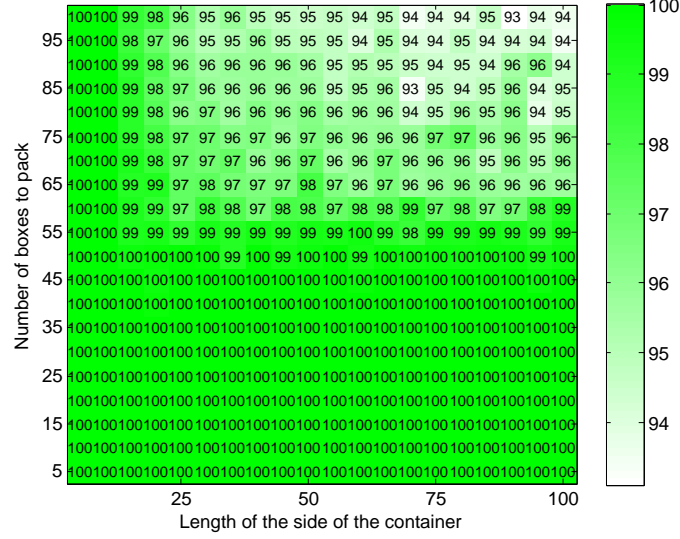


Figure 5: As above for the SAX dataset. The colours highlight the quality in terms of $100(1 - s/b)$ for solution with value s and upper bound b after one hour.

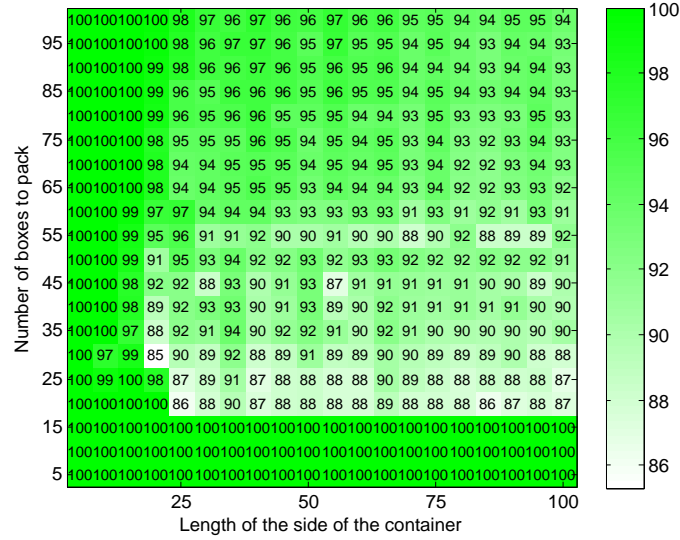


Table 3: The performance of Gurobi 4.0 on 3D Pigeon Hole Problem instances encoded in the Chen/Padberg/Fasano and the discretised formulations as reported in Allen et al. [2012]. “—” denotes that no integer solution has been found.

	Time (s)	
	Chen/Padberg/Fasano	Discretised
Pigeon-01	< 1	< 1
Pigeon-02	< 1	< 1
Pigeon-03	< 1	< 1
Pigeon-04	< 1	< 1
Pigeon-05	< 1	< 1
Pigeon-06	< 1	< 1
Pigeon-07	1.5	< 1
Pigeon-08	7.4	< 1
Pigeon-09	88.6	< 1
Pigeon-10	1381.4	< 1
Pigeon-100	—	< 1
Pigeon-1000	—	1.0
Pigeon-10000	—	1.8
Pigeon-100000	—	4.2
Pigeon-1000000	—	45.1
Pigeon-10000000	—	664.0
Pigeon-100000000	—	—

For the SA and SAX datasets, Figures 4 and 5 summarise solutions obtained within an hour using either the Chen/Padberg/Fasano formulation or the reformulation, whichever was faster. This shows that although it may be possible to solve certain instances with 10000000 boxes within an hour to optimality, real-life instances with hundreds of boxes may still be challenging, even considering the reformulation. Nevertheless, as has been pointed out by Allen et al. [2012], the space-indexed relaxation provides a particularly strong upper bound. The mean integrality gap, or the ratio of the difference between root linear programming relaxation value and optimum to optimum has been 10.49 % and 0.37 % for the Chen/Padberg/Fasano and the space-indexed formulation, respectively, on the SA and SAX instances solved to optimality within the time limit of one hour.

6 Conclusions

Overall, the discretisation provides a particularly strong relaxation, and is easy to reformulate to, programmatically, provided the constraints are a contigu-

ous block of rows, as they are, whenever the instances are produced by algebraic modelling languages. Clearly, it would be good to develop tests whether the reformulation is worthwhile, as the discretised relaxation may become prohibitively large and dense for instances with many distinct box-types, and box-types or containers of large sizes in terms of the units of discretization. Alternatively, one may consider multi-level discretisations. Plausibly, one may also apply similar structure-exploiting approaches to “components” other than packing. Mareček [2012] studied the graph colouring component, for instance. This may be open up new areas for research in computational integer programming.

Acknowledgements The code for computational testing of the performance of solvers packing problems has code developed by Allen [2012] for Allen et al. [2012] and can be downloaded at <http://discretisation.sf.net> (September 30th, 2014). This material is loosely based upon otherwise unpublished Chapter 8 of the dissertation of Mareček [2012], but has benefited greatly from the comments of two anonymous referees, which have helped the author improve both the contents and the presentation. The views expressed in this chapter are personal views of the author and should not be construed as suggestions as to the product road map of IBM products.

References

- Tobias Achterberg. SCIP: solving constraint integer programs. *Math. Program. Comput.*, 1(1):1–41, 2009. ISSN 1867-2949. doi: 10.1007/s12532-008-0001-1.
- Sam D. Allen. *Algorithms and Data Structures for Three-Dimensional Packing*. PhD thesis, University of Nottingham, 2012. URL http://etheses.nottingham.ac.uk/2779/1/thesis_nicer.pdf.
- Sam D. Allen, Edmund K. Burke, and Jakub Mareček. A space-indexed formulation of packing boxes into a larger box. *Oper. Res. Lett.*, 40:20–24, 2012. doi: 10.1016/j.orl.2011.10.008. URL <http://discretisation.sourceforge.net/space-indexed.pdf>.
- Mauro Maria Baldi, Guido Perboli, and Roberto Tadei. The three-dimensional knapsack problem with balancing constraints. *App. Math. Comput.*, 218(19):9802–9818, 2012.
- JE Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Oper. Res.*, 33(1):49–64, 1985.
- Hatem Ben Amor and José Valério de Carvalho. Cutting stock problems. *Column Generation*, pages 131–161, 2005.
- Daniel Bienstock and Mark Zuckerberg. Solving lp relaxations of large-scale precedence constrained problems. In Friedrich Eisenbrand and F. Shepherd, editors, *Integer Programming and Combinatorial Optimization*, volume 6080 of *Lecture Notes in Computer Science*, pages 1–14. Springer Berlin / Heidelberg, 2010.

- C. S. Chen, S. M. Lee, and Q. S. Shen. An analytical model for the container loading problem. *European J. Oper. Res.*, 80(1):68–76, 1995. ISSN 0377-2217. doi: 10.1016/0377-2217(94)00002-T.
- Miroslav Chlebík and Janka Chlebíková. Hardness of approximation for orthogonal rectangle packing and covering problems. *J. Discrete Algorithms*, 7(3):291–305, 2009. ISSN 1570-8667. doi: 10.1016/j.jda.2009.02.002.
- Thiago A. de Queiroz, Flavio K. Miyazawa, Yoshiko Wakabayashi, and Eduardo C. Xavier. Algorithms for 3d guillotine cutting problems: Unbounded knapsack, cutting stock and strip packing. *Comput. Oper. Res.*, 39(2):200–212, 2012. doi: 10.1016/j.cor.2011.03.011.
- G Fasano. Cargo analytical integration in space engineering: A three-dimensional packing model. In Tito A. Ciriani, Stefano Gliozzi, Ellis L. Johnson, and Roberto Tadei, editors, *Operational Research in Industry*, pages 232–246. Purdue University Press, 1999.
- Giorgio Fasano. A mip approach for some practical packing problems: Balancing constraints and tetris-like items. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 2(2):161–174, 2004.
- Giorgio Fasano. Mip-based heuristic for non-standard 3d-packing problems. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 6(3):291–310, 2008.
- Giorgio Fasano. Tetris-like items. In *Solving Non-standard Packing Problems by Global Optimization and Heuristics*, SpringerBriefs in Optimization, pages 7–26. Springer International Publishing, 2014a. ISBN 978-3-319-05004-1. doi: 10.1007/978-3-319-05005-8_2. URL http://dx.doi.org/10.1007/978-3-319-05005-8_2.
- Giorgio Fasano. Model reformulations and tightening. In *Solving Non-standard Packing Problems by Global Optimization and Heuristics*, SpringerBriefs in Optimization, pages 27–37. Springer International Publishing, 2014b. ISBN 978-3-319-05004-1. doi: 10.1007/978-3-319-05005-8_3. URL http://dx.doi.org/10.1007/978-3-319-05005-8_3.
- Giorgio Fasano. Erratum to: Chapter 3 model reformulations and tightening. In *Solving Non-standard Packing Problems by Global Optimization and Heuristics*, SpringerBriefs in Optimization, pages E1–E2. Springer International Publishing, 2014c. ISBN 978-3-319-05004-1. doi: 10.1007/978-3-319-05005-8_8. URL http://dx.doi.org/10.1007/978-3-319-05005-8_8.
- Giorgio Fasano and J Pintér. *Modeling and optimization in space engineering*. Springer, New York, NY, 2013.
- M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search algorithm for a routing and container loading problem. *Transport. Sci.*, 40(3):342–350, 2006. ISSN 0041-1655.

- PC Gilmore and RE Gomory. Multistage cutting stock problems of two and more dimensions. *Oper. Res.*, 13(1):94–120, 1965.
- JC Herz. Recursive computational procedure for two-dimensional stock cutting. *IBM J. Res. Dev.*, 16(5):462–469, 1972.
- Manuel Iori, Juan-José Salazar-González, and Daniele Vigo. An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transport. Sci.*, 41(2):253–264, 2007.
- Leonardo Junqueira, Reinaldo Morabito, and Denise Sato Yamashita. Three-dimensional container loading models with cargo stability and load bearing constraints. *Comput. Oper. Res.*, 39(1):74–85, 2012. ISSN 0305-0548. doi: 10.1016/j.cor.2010.07.017.
- Leonardo Junqueira, Reinaldo Morabito, Denise Sato Yamashita, and Horacio-Hideki Yanasse. Optimization models for the three-dimensional container loading problem with practical constraints. In Giorgio Fasano and Jnos D. Pintr, editors, *Modeling and Optimization in Space Engineering*, volume 73 of *Springer Optimization and Its Applications*, pages 271–293. Springer New York, 2013. ISBN 978-1-4614-4468-8. doi: 10.1007/978-1-4614-4469-5_12. URL http://dx.doi.org/10.1007/978-1-4614-4469-5_12.
- Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, Andrea Lodi, Hans Mittelman, Ted Ralphs, Domenico Salvagnin, Daniel E. Steffy, and Kati Wolter. MIPLIB 2010. *Math. Program. Comput.*, 3(2):103–163, 2011. doi: 10.1007/s12532-011-0025-9. URL <http://mpc.zib.de/index.php/MPC/article/view/56/28>.
- Oli BG Madsen. Glass cutting in a small firm. *Math. Program.*, 17(1):85–90, 1979.
- Jakub Mareček. *Exploiting Structure in Integer Programs*. PhD thesis, University of Nottingham, 2012. URL <http://researcher.watson.ibm.com/researcher/files/ie-jakub.marecek/dr.pdf>.
- Silvano Martello, David Pisinger, and Daniele Vigo. The three-dimensional bin packing problem. *Oper. Res.*, 48(2):256–267, 2000.
- Eduardo Moreno, Daniel Espinoza, and Marcos Goycoolea. Large-scale multi-period precedence constrained knapsack problem: A mining application. *Electronic Notes in Discrete Mathematics*, 36:407 – 414, 2010. ISSN 1571-0653. doi: 10.1016/j.endm.2010.05.052. ISCO 2010 - International Symposium on Combinatorial Optimization.
- Manfred Padberg. Packing small boxes into a big box. *Math. Methods Oper. Res.*, 52(1):1–21, 2000.

- Yunpeng Pan and Leyuan Shi. On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. *Math. Program.*, 110(3, Ser. A):543–559, 2007. ISSN 0025-5610.
- Christos H. Papadimitriou. Worst-case and probabilistic analysis of a geometric location problem. *SIAM J. Comput.*, 10(3):542–557, 1981. ISSN 0097-5397.
- Christos H. Papadimitriou. On the complexity of unique solutions. *J. Assoc. Comput. Mach.*, 31(2):392–400, 1984. ISSN 0004-5411. doi: 10.1145/62.322435.
- Jorge P. Sousa and Laurence A. Wolsey. A time indexed formulation of non-preemptive single machine scheduling problems. *Math. Program.*, 54:353–367, 1992. ISSN 0025-5610.
- Janna Magrietje van den Akker. *LP-based solution methods for single-machine scheduling problems*. Technische Universiteit Eindhoven, Eindhoven, 1994. Dissertation.
- Eitan Zemel. Probabilistic analysis of geometric location problems. *SIAM J. Algebraic Discrete Methods*, 6(2):189–200, 1985. ISSN 0196-5212. doi: 10.1137/0606017.